

Towards a more feasible quantum speedup for solving linear systems problems with Quantum Simulation

Lucas Amaral

Universidade Federal Fluminense
Rua Passo da Pátria S/N, Boa Viagem, Niterói - RJ
amarallucas@id.uff.br

Luis Kowada

Universidade Federal Fluminense
Rua Passo da Pátria S/N, Boa Viagem, Niterói - RJ
luis@ic.uff.br

ABSTRACT

Solving systems of linear equations is one of the main problems in sciences, even being already efficient classically, processing a great amount of data, as is required nowadays, can be troublesome. Quantum solutions are also being considered with this purpose. The HHL quantum algorithm aims to solve such problems with exponential *speedup* on the number of variables. In recent years, this algorithm has been established as one of the main algorithms in the recent area of Quantum Machine Learning (QML). However, with great enthusiasm regarding its practical usage and the restrictions that the algorithm requires. One restriction imposed is the efficient preparation of the coefficient matrix of the system, already assumed by the authors. In this work, we will expose these problems and offer an implementation of one of the algorithms in the literature that implements the coefficient matrix efficiently for, if it is sparse, and still gain efficiency.

KEYWORDS. Quantum Machine Learning. Linear System Problem. Quantum Simulation.

1. Introduction

In 2009, the *HHL* algorithm was presented, promising exponential time savings for solving systems of linear equations (Harrow et al. [2009]). Currently, the algorithm is considered an important tool in the framework used to solve quantum linear systems problems (*Quantum Linear Systems Problems - QLSP*).

Despite the rapid enthusiasm for the algorithm, it did not come without some skepticism. The algorithm assumes certain properties that can lead to disbelief regarding its real efficiency. Aaronson [2014] cites some difficulties in *Quantum Machine Learning (QML)* algorithms, especially *HHL*. Among the main difficulties, Aaronson points to the problem of efficiently preparing the inputs of the algorithm the Hamiltonian H .

In the following sections, we will explore these problems in a little more depth. Section 2 introduces quantum computation and the key aspects that it differs from classical computation. In Section 3, we present the *HHL* quantum circuit and the idea behind implementing an inverse matrix. In Section 4, we will present the problem of efficiently simulating quantum systems, which is apparently a necessary tool in *HHL* to encode the input matrix representing the coefficients of the system of equations. In Section 5, we present an quantum algorithm used to simulate some kind of matrices efficiently. Section 6 presents results of the execution of *HHL* using the quantum algorithm compared to the classical preparation. Section 7, we conclude the work with a discussion of the results and future research directions.

2. Quantum Computation

In quantum computation, information is represented using **quantum bits**, or **qubits**. Unlike classical bits, which can only exist in a state of 0 or 1, qubits can exist in a **superposition**, a combination of both states simultaneously. A qubit's state is described as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (1)$$

where α and β are complex amplitudes such that $|\alpha|^2 + |\beta|^2 = 1$. These squared amplitudes represent the probabilities of measuring the qubit as 0 or 1, respectively.

Multiple qubits can also be **entangled**, forming joint states that cannot be separated into individual qubit states. This creates powerful correlations that are central to quantum advantage.

Quantum computation stands apart from classical computing due to several distinctive features:

- **Superposition:** A qubit can encode multiple values at once, enabling parallelism.
- **Entanglement:** Allows qubits to be linked, enabling coordinated changes that can't be replicated classically.
- **Measurement:** The process of extracting classical information from qubits. Unlike classical systems, where reading a bit has no side effect, quantum measurement fundamentally alters the state of a system. When a qubit in superposition is measured, it **collapses** to one of the basis states— $|0\rangle$ or $|1\rangle$ —with probabilities given by the squared amplitudes of the state vector.

For example, measuring:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

will result in either 0 or 1 with equal 50% probability. After measurement, the system can no longer be in a superposition, it adopts the measured state.

This probabilistic nature of measurement means quantum algorithms must be designed to **maximize the probability** that a correct result appears upon measurement. In many cases, algorithms are repeated multiple times to statistically confirm the correct output.

The power of quantum algorithms comes from their ability to **manipulate probability amplitudes** across multiple paths at once. Rather than checking solutions one by one, quantum algorithms explore many at once, using interference to cancel wrong answers and amplify right ones.

By cleverly preparing quantum states and manipulating them through unitary operations before measurement, quantum algorithms can achieve significant speedups for specific problem classes.

3. The HHL Algorithm

The problem essentially consists of obtaining the solution of a system of linear equations in the form:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases} \quad (2)$$

The same A system of equations can also be represented in matrix form. In this case, we have, as input, a matrix, for example, A , and a vector b . Thus, we have the following representations:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} |b\rangle = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad (3)$$

The general idea of the algorithm is as follows:

Input: The input of the algorithm is a system of linear equations represented by a hermitian operator A , that is, $(A = A^\dagger)$ and a vector represented by the amplitudes of a quantum state $|b\rangle$. Since it is not likely that systems of linear equations have such a configuration, a trick is presented that relaxes this restriction. If A is not a Hermitian matrix, it is possible to transform it by expanding the matrix and the input vectors with the following trick (Harrow et al. [2009]):

$$C = \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix} \quad (4)$$

which C would be a new Hermitian matrix that could be solved in the new form $Cy = \begin{pmatrix} \vec{b} \\ 0 \end{pmatrix}$ to obtain $y = \begin{pmatrix} 0 \\ \vec{x} \end{pmatrix}$. Let us assume that A , in this work, is already Hermitian or transformed by the method above. Thus, we want to solve the system of linear equations in the form:

$$A|x\rangle = |b\rangle \quad (5)$$

We always apply quantum operators until we reach the following configuration:

$$|x\rangle = A^{-1}|b\rangle \quad (6)$$

The reason why the matrix A is represented as a Hermitian matrix is because its spectral power was represented as in 7.

$$\sum_{i=0}^{2^{n_b}-1} \lambda_i |u_i\rangle \langle u_i| \quad (7)$$

where λ_i and $|u_i\rangle$ are, respectively, the i -th eigenvalue and eigenvector of A . Thus, A is composed of a diagonal matrix of the form:

$$\begin{pmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{2^{n_b}-1} \end{pmatrix} \quad (8)$$

multiplied by the eigenvectors.

To get A^{-1} , we simply invert the eigenvalues:

$$A^{-1} = \sum_{i=0}^{2^{n_b}-1} \frac{1}{\lambda_i} |u_i\rangle \langle u_i| \quad (9)$$

Step 1: To find the inverse of A , we first use *Quantum Phase Estimation (QPE)*, an algorithm with which we can extract the eigenvalues of a matrix by encoding them in *qubits*.

Step 2: Then we apply the inversion rotations with the help of a *ancilla qubit*, this is the inversion of the eigenvalue mentioned above.

Step 3: Finally, we apply the inverse computation of *QPE*. The solution is encoded by the amplitudes of the quantum state and cannot be obtained through a measurement. However, as the authors explain, they can be used as a subroutine for other problems.

3.1. Quantum Fourier Transform

The *Quantum Fourier Transform*, or *QFT*, is a useful tool in quantum computing, since it can be performed exponentially faster quantumly. The *QFT* can be seen as a transformation of the initial state, more specifically, as a change of basis. The basis obtained as output is called the *Fourier* basis, as opposed to the computational basis, which we perform measurements on (Nielsen and Chuang [2000]).

The *QFT* transforms the orthonormal basis $|0\rangle \cdots |N-1\rangle$ into a basis resulting from the linear transformation of the following operator Nielsen and Chuang [2000]:

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k / N} |N\rangle \quad (10)$$

where the state $|j\rangle$ is the binary representation of the states of the computational basis and k are multiples of 2 that represent the position of each *qubit*.

It is convenient, for this work, to present the *QFT* from the product of each *qubit*. This representation is shown in the equation 11.

$$|j_1, \dots, j_n\rangle \rightarrow \frac{(|0\rangle + e^{2\pi i 0 \cdot j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0 \cdot j_1 \dots j_n} |1\rangle)}{2^{n/2}} \quad (11)$$

From this representation it is easy to see that the *QFT* can be applied from a sequence of rotations, where, for the first term, one rotation is applied, and for the last term, n rotations. Each rotation can be applied from the unitary gate in 12.

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i / 2^{2^k}} \end{bmatrix} \quad (12)$$

3.2. Quantum Phase Estimation

It is known, by the Spectral Theorem, that Hermitian matrices can be diagonalizable, that is, they can be decomposed into $VDV^\dagger$. Thus, from the Hermitian matrix given as input to the algorithm, we can construct a unitary U that has eigenvector $|u\rangle$ from V and eigenvalue $e^{2\pi i \varphi}$ from D (Aaronson [2016]; Nielsen and Chuang [2000]).

The construction of U and $|u\rangle$ is not handled by the procedure, therefore, the *QPE* is often classified as a subroutine. Incidentally, the preparation of the coefficient matrix and the state $|b\rangle$ for the *QLSP* itself is also delegated to an oracle assuming that there is an algorithm that constructs it efficiently.

For *QPE*, two registers are used: the first register has t qubits where t is the number of digits for the desired accuracy. An initial state $|v\rangle$ is prepared with as many qubits as necessary. The algorithm is based on the application of *Hadamard* gates of the t qubits of the first register and, then, the controlled application of the controlled gates U to each qubit raised to the power of 2. This produces $|c\rangle_t |v\rangle \rightarrow |c\rangle_t U^c |v\rangle$ (Nielsen and Chuang [2000]).

This construction, in fact, is analogous to the classical concept of eigenstate, in which, when we multiply a matrix so many times by a vector, at a certain point, there is no more rotation of the vector. In the quantum case, we apply U to each qubit raised to the power of two according to the binary place of each digit.

The final state is given by the equation 13.

$$\frac{1}{2^{t/2}} (|0\rangle + e^{2\pi i 2^{t-1} \varphi} |1\rangle) \dots (|0\rangle + e^{2\pi i 2^0 \varphi} |1\rangle) |v\rangle \quad (13)$$

Note that each exponent 2^i refers to the binary place (analogous to a base-2 decimal place) of φ and that the state $|v\rangle$ remained the same throughout the procedure.

The important thing to note, to complete the procedure, is that, by comparing 11 and 13, we can extract $|\varphi_0 \varphi_1 \dots \varphi_t\rangle$ by applying *QFT* after *QPE*. In this way, we obtain the eigenvalues of U encoded in the first register.

3.3. Controlled Rotations

At this point, we use the auxiliary qubit to apply the rotations needed to invert the eigenvalues (Harrow et al. [2009]). To rotate the eigenvalues, we apply multi-controlled $RY(\theta)$ so that the eigenvalues are encoded by the qubits. The $RY(\theta)$ gate is shown in the equation 14.

$$RY(\theta) = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix} \quad (14)$$

What we aim for is the following transformation:

$$|0\rangle \rightarrow \sqrt{1 - \frac{1}{\lambda_j^2}} |0\rangle + \frac{1}{\lambda_j} |1\rangle \quad (15)$$

From this equation, we can see that if the auxiliary *qubit* is $|1\rangle$, we obtain an eigenvalue inversion. To prepare this state, we define $\theta = 2 \arcsin \frac{1}{\lambda}$.

After these steps, we apply the inverse computation (*uncomputation*). We use QPE^{-1} to obtain the result of the multiplication between $\lambda_i^{-1} |u_i\rangle \langle u_i|$ in the first register, and $|b\rangle$ encoded in the second, resulting in $|x\rangle$ in the second register.

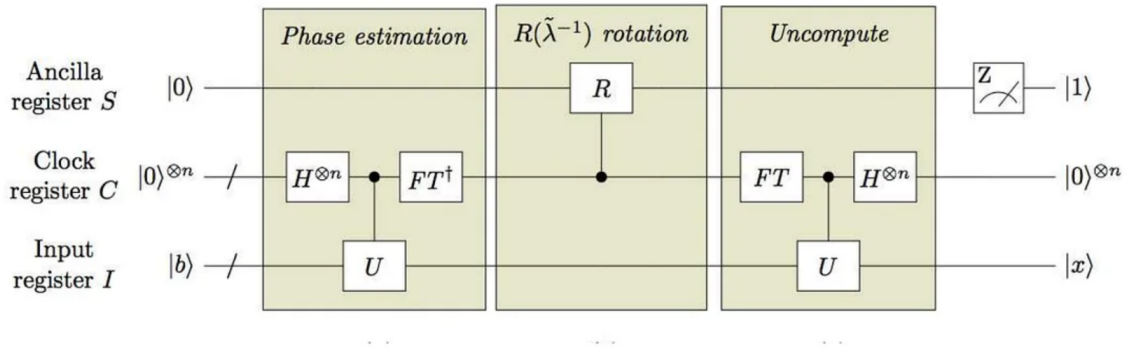


Figure 1: HHL circuit.

4. Simulating Hamilton Dynamics

The goal of simulating dynamical systems is, given an initial state of the system, to discover the state at another time or position. The solution is obtained from the approximation of the initial state and then the application of a procedure in an iterative manner to calculate the state at another time or space. The evolution of quantum systems is also given from a differential equation, the Schrödinger equation (Nielsen and Chuang [2000]):

$$i\hbar \frac{d}{dx} |\psi\rangle = H |\psi\rangle$$

where $\hbar$ is Planck's constant. For convenience, $\hbar = 1$.

The problem of simulating quantum systems is precisely the number of differential equations that need to be solved. For n qubits we have to solve 2^n differential equations which means that the number of equations grows exponentially with the number of qubits. This becomes complexly intractable since, to compute a system of only 50 qubits, we have to have the processing power of a supercomputer.

Thus, in this Chapter we will explore some techniques for simulating physical systems. Of course, we have to explore efficient ways of simulating since we are trying to verify the possibility of exploring this simulation as part of the *HHL* which claims a complexity of $\mathcal{O}(\log n)$. In Section 4.1, we will briefly explain the concept of what the Hamiltonian is and how it can be represented. In Section 4.2, we will explain the problem of simulating physical systems.

4.1. The Hamiltonian

In quantum mechanics, we represent procedures using unitary matrices. Unitary matrices are linear transformations of a discrete-time quantum state. The evolution of the system can be represented as:

$$|\psi\rangle \rightarrow U |\psi\rangle$$

In physics, however, time is treated as a continuous aspect, that is, instead of jumping from a state $|\psi\rangle$ to a state $U |\psi\rangle$, this "jump" is treated as a continuous process performed in a time

interval. Thus, we define the **Hamiltonian** as a generator of unitary operators from an instantaneous time. Hamiltonians are always represented as Hermitian matrices, that is, given a Hamiltonian H , we have that $H = H^\dagger$. The Hamiltonian is not necessarily a positive semi-definite matrix, that is, it can have negative eigenvalues (Aaronson [2016]).

As already mentioned, the equation that describes the evolution of a quantum system is the Schrödinger equation:

$$i \frac{d}{dt} |\psi\rangle = H |\psi\rangle$$

where H is a Hamiltonian. Assuming that H is independent of time, we can derive from the equation that the state of the system after time t is:

$$|\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle$$

In this way, we transform a differential equation which we would solve for each coordinate of the vector $|\psi\rangle$ into an equation in which we exponentiate a matrix as if it were a scalar (Aaronson [2016]).

Exponentiating a matrix may be an unusual concept, but we can verify that, for any diagonal matrix, the solution is the exponentiation of the diagonal terms. That is:

$$\exp\left(\begin{bmatrix} \lambda_0 & & \\ & \ddots & \\ & & \lambda_{n-1} \end{bmatrix}\right) = \begin{bmatrix} e^{\lambda_0} & & \\ & \ddots & \\ & & e^{\lambda_{n-1}} \end{bmatrix}$$

So the problem becomes representing our Hamiltonian as a diagonal matrix. We know that any Hermitian matrix can be decomposed into $VDV^\dagger$ by the Spectral Theorem. Thus we can write e^A as $Ue^DU^\dagger$. So to exponentiate the matrix, it needs to be decomposed (Aaronson [2016]).

However, in the context of *HHL*, decomposing the matrix to encode the Hamiltonian becomes infeasible since we aim for a complexity of $\mathcal{O}(\log n)$. How can we improve this? Are there methods to simulate this Hamiltonian efficiently?

4.2. The Simulation Problem

So, as seen in the previous Section, we represent the dynamics of a quantum system from the Schrödinger equation and the solution of the system, given a time-independent Hermitian matrix, is $e^{-iHt} |\psi_{\text{initial}}\rangle$.

$$i\hbar \frac{d}{dt} |\psi\rangle = H |\psi\rangle \quad \Rightarrow \quad |\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle$$

The problem is that exponentiating a matrix like e^{-iHt} can be computationally difficult, even though the matrix is sparse the calculation can be exponential. Only decomposing the matrix utilizing numerical methods cost $\mathcal{O}(m^3)$. However, there are instances where the Hamiltonian can be written as a linear combination of local terms where each term acts on part of the system (Nielsen and Chuang [2000]; Lloyd [1996]):

$$H = \sum_{l=1}^L H_l$$

The Hamiltonian is called k -local when H acts on m qubits and each H_l acts, nontrivially, on at most k qubits. It is assumed that each H_l is also Hermitian and can be directly exponentiated, that is, it is an evolution that can be constructed as a quantum circuit and executed on a quantum

computer Lloyd [1996]. Physically, this concept is also very important since most Hamiltonians occurring in nature are k -local.

Hence it is possible to conclude that it is possible to decompose local Hamiltonians into the evolution of the complete Hamiltonian into the product of the evolutions of the local terms. Furthermore, since each exponent can be represented as a quantum circuit, the problem is solved for the entire system.

$$e^{-iHt} = e^{-iH_1t} e^{-iH_2t} \dots e^{-iH_Lt}$$

$$\text{if } [H_i, H_j] = 0$$

The restriction $[H_i, H_j] = H_i H_j - H_j H_i = 0$, that is, if all local terms commute with each other. This occurs because this equation is derived from the binomial formula that generates products such as $H_1 H_2 H_1$ which is different from $H_1^2 H_2$.

Thus, the problem of simulating a Hamiltonian can be described as follows:

Hamiltonian Simulation Problem: Given a Hamiltonian H , a square Hermitian matrix $2^n \times 2^n$, acting on n qubits in time t and error ϵ , the objective is to find the sequence of quantum gates that implement an evolution as a function of time U whose norm of the difference between the simulation and the ideal evolution is, at most, ϵ . In other words:

$$\|U - e^{iHt}\| \leq \epsilon \quad (16)$$

If $[H_i, H_j] \neq 0$, the product of local terms does not hold. In this case, another tool shown below is used.

$$e^{A+B} = \lim_{m \rightarrow \infty} (e^{A/m} e^{B/m})^m \quad (17)$$

in this case, m is the number of *trotter steps*. This value should be assigned aiming at the number of iterations necessary for the simulation error to reach the desired value of ϵ , given in 16. A and B are Hermitian matrices that act on k qubits with $k < n$. In this way, the Hamiltonian simulation of a matrix can be performed, from the decomposition into local Hamiltonians A and B , in m iterations. It is important to note that there are variations of 17, here the 1st order *trotter formula* is shown.

5. Simulating Sparse Matrices

Sparse Hamiltonians naturally arise in physical systems with local interactions, such as spin chains and lattice field theories, making their efficient simulation a key challenge for quantum algorithms. So simulating such systems became an important problem in quantum computation.

The theoretical groundwork was laid by Lloyd [1996], who demonstrated that local Hamiltonians could be simulated efficiently on a quantum computer using *Trotter-Suzuki product formulas*. In this approach, the time evolution operator e^{-iHt} is approximated by a product of exponentials of simpler, often local, components $H = \sum_i H_i$.

In 2003, Aharonov and Ta-Shma [2003] extended the scope of Hamiltonian simulation to *sparse* matrices. They formalized the notion of *efficiently row-computable* Hamiltonians matrices for which the nonzero entries and their indices can be computed efficiently and showed that such systems could be simulated in polynomial time on a quantum computer, assuming oracle access to the matrix structure. This broadened the class of quantum simulable systems beyond strictly local Hamiltonians.

In 2007, a pivotal contribution, Berry, Ahokas, Cleve, and Sanders introduced refined high-order *Trotter-Suzuki formulas* that significantly reduced the gate complexity required for simulating sparse Hamiltonians (Berry et al. [2007]).

5.1. The Coloring Algorithm

A key technique introduced by Berry et al. [2007] for simulating sparse Hamiltonians involves decomposing a d -sparse Hermitian matrix $H \in \mathbb{C}^{N \times N}$ into a sum of efficiently simulatable 1-sparse matrices. This is accomplished through a graph-based *coloring algorithm*, which ensures that each resulting component can be simulated independently using efficient quantum subroutines.

Step 1 - Graph Representation To begin, the sparse matrix H is interpreted as the adjacency matrix of an undirected graph $G_H = (V, E)$, where each vertex $v \in V$ corresponds to a basis state, and an edge exists between u and v if $H_{uv} \neq 0$. Because H is Hermitian, the graph is undirected: $H_{uv} \neq 0 \Rightarrow H_{vu} \neq 0$.

Step 2 - Oracle Access The method assumes access to two quantum oracles:

- A **position oracle** O_f , which returns the position of the j -th nonzero entry in row v .
- A **value oracle** O_H , which provides the corresponding matrix element $H_{v,j}$.

These oracles allow the algorithm to access the sparse structure of H without needing the full matrix.

Step 3 - Edge Labeling and Color Assignment To enable decomposition, the algorithm assigns *colors* to the edges of the graph such that no two edges of the same color share a vertex. Each edge (u, v) is labeled using a deterministic, symmetric function of the indices of u and v in each other's neighbor lists. Specifically, let:

- i be the index of v in the list of u 's neighbors.
- j be the index of u in the list of v 's neighbors.

The **color** assigned to the edge is then the pair (i, j) . Because both i and j range from 1 to d , the number of distinct colors is bounded above by d^2 . Importantly, this guarantees that each color class corresponds to a subgraph where each vertex has degree at most one, that is, a 1-regular subgraph.

To ensure consistency in determining the index i or j (especially if neighbor ordering is ambiguous), Berry et al. employ a technique called **deterministic coin tossing** Cole and Vishkin [1986]. This is a symmetry-breaking method that allows two vertices to consistently determine which one is "greater" using a fixed rule based on their binary identifiers. The procedure allows for a total order to be derived locally, without randomness, by comparing the bit representations of vertex labels lexicographically.

This approach ensures that both endpoints of an edge (u, v) can independently compute the same color assignment using only local information and oracle access to their neighbors, maintaining coherence in the color decomposition across the entire matrix.

Step 4 - Decomposition into 1-Sparse Matrices Each color class defines a **1-sparse Hermitian matrix** $H_{i,j}$, where:

$$(H_{i,j})_{uv} = \begin{cases} H_{uv}, & \text{if edge } (u, v) \text{ has color } (i, j), \\ 0, & \text{otherwise.} \end{cases}$$

Thus, the original Hamiltonian can be written as:

$$H = \sum_{i=1}^d \sum_{j=1}^d H_{i,j},$$

with each $H_{i,j}$ containing at most one nonzero entry per row and column, and therefore efficiently simulatable. The 1-sparse matrix can be efficiently computed with the following circuit, also making use of the oracles O_f and O_h defined earlier.

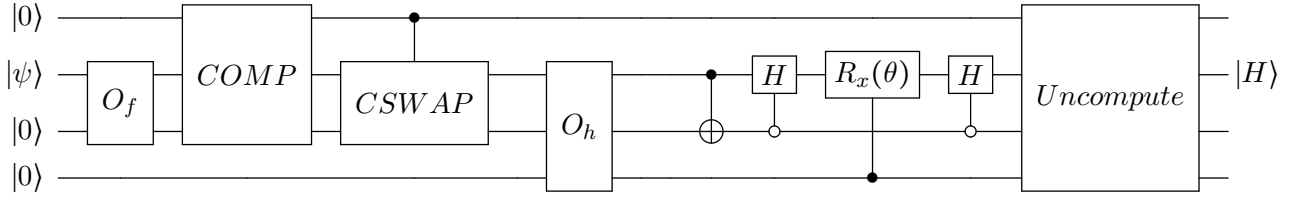


Figure 2: Quantum circuit for simulating 1-sparse matrices.

The simulation proceeds by sequentially applying approximations of the time-evolution operator $e^{-iH_{i,j}t}$ for each 1-sparse component. The complete simulation of e^{-iHt} is then achieved using a Trotter-Suzuki or higher-order decomposition scheme over the sum of all $H_{i,j}$.

This coloring algorithm is both *constructive* and *efficient*, requiring only local access to the matrix entries and their positions. It plays a central role in sparse Hamiltonian simulation algorithms by reducing the complex problem of simulating a d sparse matrix to that of simulating $\mathcal{O}(d^2)$ much simpler 1-sparse matrices, each of which admits known efficient quantum implementations.

6. Results

To assess the precision of sparse Hamiltonian simulation via the coloring algorithm, we implemented the Harrow-Hassidim-Lloyd (HHL) quantum algorithm on a 4×4 Hermitian matrix A . The unitary $U = e^{iAt}$ was simulated using two approaches:

1. **Sparse Simulation via Coloring:** The matrix A was decomposed into a sum of 1-sparse matrices using the coloring algorithm, and exponentiated using a first-order Trotter-Suzuki expansion.
2. **Exact Diagonalization:** The matrix was diagonalized classically as $A = VDV^\dagger$, enabling an exact construction of $U = Ve^{iDt}V^\dagger$.

6.1. Fidelity Comparisons by Trotter Step Count

We ran the HHL circuit for each version of U , measuring the output quantum state in 1000 shots. The comparison was repeated using three different Trotter step counts: 1, 50, and 100. The goal was to evaluate whether the algorithm actually approximates to the coefficient matrix and if the approximation relates to the number of *trotter steps*.

We ran the algorithm on Google Colab on a virtual machine with 12 gigabytes of RAM. We used the Qiskit Version 3 package to run the algorithms on simulators. As the resources of the classical and quantum machine are limited, we chose a small dimensional 4 by 4 2-sparse matrix as an example.

$$H = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 3 & 0 & 0 \\ 0 & 0 & 3 & 4 \\ 1 & 0 & 4 & 0 \end{bmatrix} \quad (18)$$

Figures 3, ??, and ?? show the results of the HHL simulations using the sparse matrix simulation with Trotter steps 1, 50, and 100 respectively, compared to the ideal result using the $VDV^\dagger$ method.

HHL output comparison using sparse simulation exact diagonalization vs (trotter step = 1, 50, 100).

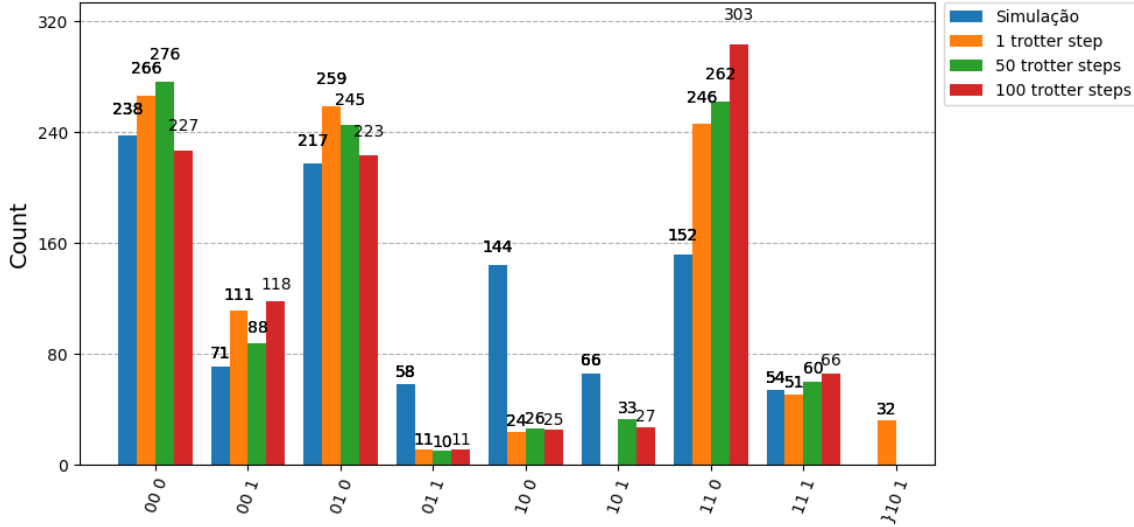


Figure 3: HHL output comparison using sparse simulation exact diagonalization vs (trotter step = 1, 50 and 100).

Trotter steps	Fidelity
1	0.82
50	0.91
100	0.89

Figure 4: Fidelity for 1, 50 and 100 trotter steps.

From the count results retrieved by these executions, the fidelity can be calculated for each case. The fidelity can be seen as the difference between the exact expected solution that we obtained by simulating classically and the approximated quantum solution. The less the fidelity is, more the result approximates the expected solution, see Figure 4.

$$\sum \sqrt{(p_i q_i)^2}$$

where p and q represents the quantum states we want to know the "closeness". The results clearly illustrate the trade-off between simulation fidelity and Trotter step count. With a single Trotter step, the approximation error in simulating e^{iAt} causes a noticeable deviation in the HHL result. As the number of Trotter steps increases to 50 and 100, the fidelity improves substantially, closely matching the output of the exact diagonalization method but stops getting significant increase from 50 to 100 trotter steps. Also, we see a difference in the pattern of the counts from the trotter executions in relation to the classical decomposition. We would expect that this difference would vanish as we increase the number of trotter steps but further investigation may be needed.

7. Conclusion

In this work, we implemented and evaluated a quantum simulation strategy for sparse Hermitian matrices based on the coloring algorithm, as introduced by Berry et al. (2007). The simulated time evolution operator $U = e^{iAt}$ was constructed using a Trotterized decomposition of 1-sparse Hamiltonians derived from edge coloring. We integrate this simulation approach into the Harrow-Hassidim-Lloyd (HHL) algorithm to solve a linear system of equations involving a matrix 4×4 .

Our results demonstrate that increasing the number of Trotter steps significantly improves the fidelity of the HHL output state compared to the ideal result obtained from classical diagonalization $A = VDV^\dagger$.

The quantum sparse decomposition using the coloring approach operates in $\mathcal{O}(n \log n)$ time for d -sparse matrices, representing a significant improvement over the classical Gauss-Jordan elimination, which scales as $\mathcal{O}(n^3)$. This highlights the potential of quantum algorithms to efficiently handle structured linear algebra problems, especially in regimes where sparsity and Hamiltonian locality can be exploited.

Future work includes the implementation of alternative quantum simulation techniques. *Linear Combinations of Unitaries (LCU)* provide a promising route for simulating non-sparse or dense Hamiltonians with improved precision bounds, *Quantum Walk-based* simulation algorithms offer structured evolution models particularly well suited for graph-based or combinatorial problems. Comparing these methods with the coloring-based sparse simulation, both in terms of fidelity and resource requirements, will further clarify the trade-offs among different simulation strategies in realistic quantum hardware environments.

References

- Aaronson, S. (2014). Quantum machine learning algorithms: Read the fine print. *Nature Physics*, p. 5.
- Aaronson, S. (2016). Introduction to quantum information science: lecture notes.
- Aharonov, D. and Ta-Shma, A. (2003). Adiabatic quantum state generation and statistical zero knowledge. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, p. 20â29, New York, NY, USA. Association for Computing Machinery. ISBN 1581136749. URL <https://doi.org/10.1145/780542.780546>.
- Berry, D. W., Ahokas, G., Cleve, R., and Sanders, B. C. (2007). Efficient quantum algorithms for simulating sparse hamiltonians. *Communications in Mathematical Physics*, 270:359–371.
- Cole, R. and Vishkin, U. (1986). Deterministic coin tossing with applications to optimal parallel list ranking. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, p. 206–216. ACM.
- Harrow, A. W., Hassidim, A., and Lloyd, S. (2009). Quantum Algorithm for Linear Systems of Equations. *Phys. Rev. Lett.*, 103(15):150502.
- Lloyd, S. (1996). Universal quantum simulators. *Science*, 273(5278):1073–1078. URL <https://www.science.org/doi/abs/10.1126/science.273.5278.1073>.
- Nielsen, M. A. and Chuang, I. L. (2000). *Quantum Computation and Quantum Information*. Cambridge University Press.